FreshnessService

September 4, 2026

FreshnessService

Kind: Service

Source: `atloria-monorepo/apps/api/src/technical-docs/freshness.service.ts`

`FreshnessService` evaluates the freshness of technical documentation and identifies content that may be outdated. It exposes `stale()` to retrieve stale-document results and `reshoot()` to refresh or regenerate the affected documentation. The service is intended for backend workflows such as scheduled maintenance jobs, administrative actions, or documentation health checks.

Methods

Method	Signature	Returns	Description
`stale`	`stale(projectId: string, organizationId: string)`	`Promise<StaleResult>`	List the project's stale pages from technical_snapshots.staleInfo.
`reshoot`	`reshoot(projectId: string, user: JwtPayload)`	`Promise<ReshootResult>`	One-click screenshot re-shoot for the stale manual pages.

Dependencies

- `PrismaService`
- `DocAutomationService`

Where it refuses work

- `FreshnessService` stops the work with `ForbiddenException` when `!project` — “Project not found in your organization.”.

- `FreshnessService` stops the work with an early return when `!info || !manualPages.length`.
- `FreshnessService` stops the work with an early return when `!pages.length`.
- `FreshnessService` stops the work with an early return when `screenSlugs.size === 0`.

Diagram

```
sequenceDiagram
  participant Client as Caller / Job
  participant Service as FreshnessService
  participant Docs as Documentation Store

  Client->>Service: stale()
  Service->>Docs: Inspect document freshness
  Docs-->>Service: Freshness data
  Service-->>Client: StaleResult

  Client->>Service: reshoot()
  Service->>Docs: Refresh stale documentation
  Docs-->>Service: Updated document data
  Service-->>Client: ReshootResult
```

Usage

```
import { Injectable } from '@nestjs/common';
import { FreshnessService } from './technical-docs/freshness.service';

@Injectable()
export class DocumentationMaintenanceJob {
  constructor(private readonly freshnessService: FreshnessService) {}

  async run(): Promise<void> {
    const staleResult = await this.freshnessService.stale();

    // Trigger a refresh workflow when stale documentation is detected.
    if (staleResult) {
      const reshootResult = await this.freshnessService.reshoot();
      console.log('Documentation refresh completed:', reshootResult);
    }
  }
}
```

AI Coding Instructions

- Inject `FreshnessService` through NestJS dependency injection; do not instantiate it directly.
- Call `stale()` before `reshoot()` when implementing maintenance workflows so refresh operations are driven by freshness state.
- Preserve the `Promise<StaleResult>` and `Promise<ReshootResult>` contracts when extending service behavior.
- Keep callers responsible for scheduling, authorization, and presentation; keep freshness evaluation and refresh logic inside this service.
- Handle service errors at the integration boundary, especially for scheduled jobs or administrative endpoints.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `DocAutomationService`

Referenced By

- `FreshnessController` (DEPENDS_ON)
- `TechnicalDocsModule` (MODULE_PROVIDES)